**Commandes en JavaScript**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Exemple | Utilisation | |
| Commandes |  |  |  |  | |
| \n | dans une chaine = retour à la ligne | "Txt\ntxtALaLigne" |  |  |  |
| ; | clos une instruction | instruction1;  instruction2; |  |  |  |
| console.log( ); | Affiche la variable dans la console du navigateur | console.log(valeur); |  |  |  |
| // | précede une ligne de commentaire pour qu’elle soit ignorée lors de l’exécution | //Commentaire sur le code |  |  |  |
| /\* \*/ | encadre plusieurs lignes de commentaires ou portion de code ignoré | /\*Début du commentaire  ou du code ignoré  jusqu’à cette ligne\*/ |  |  |  |
| var | permet de déclarer   * variable * tableau * objet | var maVariable;  var maVariable = 3;  var monTableau = [valeur1, valeur2, valeur3];  var monObjet = {  prop1: valeurProp1,  prop2: valeurProp2,  }; |  | var maVariable1, maVariable2 = valeur, maVariable3;  maVariable1 = maVariable3 = valeur; | déclarer plusieurs variables (avec ou non une valeur) en 1 seule ligne  attribuer la même valeur à plusieurs variables en 1 seule ligne |
| ++ | incrémente une variable de type nombre d’une unité | maVariable++; | a = 3;  a++; => a = 4 |  |  |
| +=  -= | augmente ou diminue une variable de type nombre de la valeur indiquée | maVariable += 3;  maVariable -= 4; | a = 3;  a += 3 => a = 6  a -= 4 => a = -1 | var message = "Bonjour ", nom = "Alex";  message += nom | Remarque : += fonctionne aussi pour la concaténation |
| \*=  /= | remplace la valeur de la variable en la multipliant ou divisant par le nb indiqué | maVariable \*= 3;  maVariable /= 4; |  |  |  |
| %= | remplace la valeur de la variable par le reste de sa division euclidienne par le nb indiqué (modulo) | maVarible %= 3; | a = 8;  a %= 3 => a = 2 |  |  |
| ( ).  (function() {    })() | utilisation de l'élément comme un objet, permet accéder à ses méthodes et attributs  permet d'isoler une fonction et de la lancer immédiatement (l'isolation évite les conflits en cas de définition de variables identiques par plusieurs fonctions) | (12).toString()  ("toto").lengh  (function maFonction() {  instruction(s);  })() | "12"  4  /*/version anonyme :*  (function () {  instruction(s);  })() | Rq: qu'elle soit nommée ou anonyme, la fonction est appelée immédiatement ! |  |
| Number( ) | convertie un variable de type chaine en nombre | Number(maVariableChaine) | Number("5") => 5 | var nb = Number(prompt("Entrez un nombre : ")); | Permet de re-transformer une variable de type chaine donnée pas prompt() en variable de type nombre. |
| String( ) | convertie une variable de type nombre en chaine | String(maVariableNombre) | String(10) => "10" |  |  |
| typeof | renvoi le type d'une variable | typeof maVariable |  | valeurs possibles : number, string, boolean, object, undefined | si renvoi undefined : la variable n'existe pas ou n'a pas encore de valeur attribuée |
| isNaN( ) | vérifie si la variable contient NaN (= Not a Number) et renvoi true si c'est le cas | isNaN(maVariable); |  |  | permet de vérifier si la variable devait enregistrer un nb et qu'une erreur s'est produite |
| isInfinite( ) | idem avec un infinie (positif ou négatif) | isInfinite(maVariable); |  |  |  |
| Boites de dialogue |  |  |  |  |  |
| alert( ) | affiche un message d'alerte dans une boite de dialogue | alert("Txt à afficher"); |  |  |  |
| confirm( ) | affiche un message avec demande de confirmation de l'utilisateur  si OK => renvoie true  si Annulé => renvoi false | confirm("Txt à afficher"); |  | if (confirm("txt à afficher")) {  instruction(s);  } | peut s'utiliser not. dans une condition de type if |
| prompt( ) | propose à l’utilisateur d’entrer une valeur tout en lui affichant une indication | valeurUser = prompt("Txt à afficher"); | var age = prompt("Entrez votre nom :"); |  |  |
| Logique |  |  |  |  |  |
| true | valeur de type booléen : vrai | true |  |  | peut servir à comparer une expression dont le résultat est de type booléen |
| false | valeur de type booléen : faux | false |  |  | Idem |
| === | égal à  comparaison de la valeur et du type | maValeur === valeurRef | var a = 3;  a === 5 => false  a === 3 => true  a === "3" => false |  |  |
| == | égal à  comparaison de la valeur uniquement | maValeur == valeurRef | var a = 3;  a === 5 => false  a === 3 => true  a === "3" => true |  |  |
| !== | différent de  comparaison de la valeur et du type | maValeur !== valeurRef | if (a !== 3) {  instruction;  } | while (mot !== "stop") {  prompt("Entrez un mot");  } |  |
| != | différent de  comparaison de la valeur uniquement | maValeur != valeurRef |  |  |  |
| <  <=  >  >= | inférieur à  inférieur ou égal à  supérieur à  supérieur ou égal à | maValeur <= valeurRef |  |  |  |
| && | opérateur logique ET | (condition1) && (condition2) |  | if ((nb >= 0) && (nb <= 100)) | permet de savoir si |
| | | | opérateur logique OU | (condition1) || (condition2) |  | if ((nb < 0) || (nb > 100))  alert(variableA || variableB || variableC); | permet de savoir si  retourne la valeur de la première variable qui est "vrai" (true) |
| ! | opérateur logique NON | !(condition) |  | if ( !(nb < 100)) {  console.log(nb + " est supérieur ou égal à 100");  } |  |
| Conditions |  |  |  |  |  |
| if | exécute la/les instruction(s) qui suivent si la condition quelle contient est vérifiée | if (condition) instruction;  if (condition) {  instruction1;  instruction2;  } |  |  |  |
| else | sivant la comande if, exécute la serie d’instruction si la condition dans if n’est pas vérifiée | if (condition) {  instructions;  } else {  instructions;  } |  |  |  |
| switch ( )  case  default | compare la valeur d’une variable à plusieurs choix possibles (mais discrets) | switch (maVariable) {  case valeurRef1: instruction1;  break;  case valeurRef2: instruction2;  break;  default: instructionDeDéfaut;  } | default permet de donner une instruction au cas où la variable aurait une valeur différente des cas possibles listés. |  | QCM |
| ? : | condition ternaire  si la condition est vérifiée (true) alors elle renvoie la valeur1  sinon (false) elle renvoie la valeur2 | condition ? valeur1 : valeur2; |  | var startMessage = "Votre catégorie : ",  endMessage,  adult = confirm("Etes-vous majeur ?");  endMessage = adult ? "18+" : "-18";  alert(startMessage + endMessage); |  |
| Boucles |  |  |  |  |  |
| while ( ) | boucle permettant la répétition des instructions tant que la condition entre () est vérifiée | while (condition) {  instruction1;  instruction2;  } | ATTENTION de ne pas créer de boucle infinie ! | do {  instruction(s);  } while (condition); | Remarque : une boucle Do While permet d'éxécuter les instructions de la boucles au moins une fois. La condition n'est testé qu'à partir de la 2nd fois. |
| for ( ) | boucle permettant la répétition des instructions le nombre de fois voulu | for (var i = 0; condition; i++) {  instructions;  } |  |  |  |
| break; | permet de stopper et de sortir de la boucle |  |  |  |  |
| continue; | permet d'interrompre la progression de l'itération en cours et de passer à la suivante (stoppe l'itération mais pas la boucle) |  |  |  |  |
| Fonction |  |  |  |  |  |
| function | déclare une nouvelle fonction | function maFonction(param1, param2) {  instructions;  } |  | function(param) {  instruction(s);  } | cas d'une fonction anonyme |
|  | appeler une fonction en lui donnant les valeurs (arguments) à utiliser pour ses paramètres | maFonction(arg1, arg2); |  | (function (param1) {  instruction(s);  })(valeur1); | déclaration et appel immédiat d'une fonction anonyme isolée (IIFE) |
| return | permet à une fonction de renvoyer une valeur | function maFonction(p1, p2) {  var valeurRetour;  instructions;  return valeurRetour;  } | Doit être utilisé à la fin du corps de la fonction.  Une seule valeur de retour par fonction. |  |  |
| arguments | une fonction peut être appelée avec un nombre variable d'arguments (+ ou - que le nombre normal défini dans la déclaration de la fonction)  arguments = tableau contenant les arguments passés en paramètre lors de l'appel de la fonction | function maFonction(arg1, arg2) {  arguments.lenght;  arguments.forEach(function(arg) {  instruction(s);  });  } |  |  |  |
| Chaine et Tableau |  |  |  |  |  |
| .length | permet d’obtenir la taille d’une chaine ou d’un tableau | maChaine.length;  monTableau.length; |  |  |  |
| [ ] | permet d’obtenir le ne  caractère d’une chaine  ou valeur d’un tableau | maChaine[indice];  monTableau[indice]; | ATTENTION : dans une chaine ou un tableau,   * 1er indice est 0 * le dernier est taille-1 | var mot = "…"  for (var i = 0; i < mot.length; i++) {  console.log(mot[i]);  } | parcours et affiche tous les caractères d’une chaine l’un après l’autre. |
| Chaines |  |  |  |  |  |
| .charAt( ) | permet d’obtenir le ne caractère d’une chaine | maChaine.charAt(indice); |  |  |  |
| .toLowerCase( ) | convertie tous les caractères d’une chaine en minuscule | maChaine.toLowerCase(); | ATTENTION : ceci crée une nouvelle chaine. | var motSaisie = "Quitter";  While (motSaisie.toLowerCase() !== "quitter") {  …  } | Comparaison de chaines (car JavaScript est Case sensitive, cad différencie Maj et min.) |
| .toUpperCase( ) | convertie tous les caractères d’une chaine en majuscule | maChaine.toUpperCase(); | ATTENTION : ceci crée une nouvelle chaine. |  |  |
| .split( ) | sépare une chaine de caractère selon le un séparateur donné en paramètre  (et crée un tableau avec les sous chaines obtenues) | var monTableau = maChaine.split("séparateur"); |  |  |  |
| .trim() | supprime les espaces en début et fin de chaine | maChaine.trim( ); |  |  |  |
| .indexOf( ) | recherche la présence d'une chaine donnée en paramètre dans la chaine qui le précède et retourne l'index de la position de la 1re lettre dans celle-ci  retourne -1 si pas trouvée | maLongueChaine.indexOf(maChaineAChercher); |  | .lastIndexOf( ) | même principe mais retourne l'index du dernier caractère |
| .substring( , ) | extrait le contenu de la chaine entre  param1 : index inclus  param2 : index exclu | maChaine.substring(nb, nb); |  | .substr(nb, nb); | idem avec  param1 : index inclus  param2 : nb de caractère à extraire |
| Tableaux |  |  |  |  |  |
| .forEach( ) | permet de parcourir l’une après l’autre toutes les valeurs d’un tableau  elle prend 2 param. :   * la fonction qui traitera les éléments contenu dans le tableau un par un * l'objet qui servira de référence pour this utilisé dans la fonction si besoin (param facultatif) | monTableau.forEach( );  monTableau.forEach(fonction, objet); |  | monTableau.forEach(function (valeur) {  console.log(valeur);  });  monTableau.forEach(function (valeur, index, tableau) {  instruction(s);  }); | parcours et affiche toutes les valeurs du tableau l’une après l’autre  la fonction dans .forEach( ) pent prendre 3 param. :   * la valeur de la case actuelle * l'indice de la case actuelle (facult) * le tableau actuellement traité (facult) |
| .push( ) | ajoute une nouvelle valeur à la fin d'un tableau | monTableau.push(valeurN+1); |  |  |  |
| .unshift( ) | ajoute une nouvelle valeur au début du tableau | monTableau.unshift(valeur0); |  |  |  |
| .pop() | retire le dernier élément du tableau | monTableau.pop(); |  |  |  |
| .shift() | retire le premier élément du tableau | monTableau.shift(); |  |  |  |
| .join( ) | créé une chaine à partir des éléments d'un tableau en :  les séparant par l'élément en paramètre  les colants | monTableau.join(" - ");  monTableau.join(); | var maChaine = monTableau.join(" "); |  |  |
| .concat( ) | créé un nouveau tableau qui est la concaténation de deux tableaux | monGrandTableau = tableau1.concat(tableau2); |  |  |  |
| .indexOf( ) | recherche la présence d'un élément (chaine, nombre, objet, …) donnée en paramètre dans les éléments du tableau et retourne l'index de sa position  retourne -1 si pas trouvée | monTableau.indexOf(monElement); |  |  |  |
| .reverse() | inverse l'ordre des éléments d'un tableau | monTableau.reverse(); |  |  | Remarque : cette méthode ne créé pas un nouveau tableau, elle le modifie |
| .sort( ) | range les données d'un tableau par ordre alphabétique  avec une fonction comme param. facultatif pour gérer le rangement | monTableau.sort();  monTableau.sort(function(a, b) {  instruction(s);  }); |  |  | la fonction (facultative) compare les valeurs 2 à 2   * si veut conserver l'ordre : return -1; * si veut inverser a et b : return 1; * si a et b sont identique : return 0; |
| .slice( , ) | extrait une partie du tableau   * param1 : indice de départ inclus * param2 : indice de fin exclu (facult.) | monTableauExtrait = monTableau.slice(nb, nb); |  |  |  |
| .splice( , ) | supprime / remplace / insère des éléments dans un tableau   * param1 : indice à partir duquel faire les modifs * param2 : nb de case à retirer (= 0 si insertion) * param3,4,… : éléments à remplacer / insérer (facultatif si supprimer) | monTableau.splice(nb, nb);  monTableau.splice(nb, nb, valeur, valeur); |  |  |  |
|  |  |  |  |  |  |
| Objets |  |  |  |  |  |
| new | permet l’instanciation, cad de créer un objet à partir d’un constructeur (= fonction ayant pour rôle d’initialiser un objet) | function MonObjet(param1, param2) {  this.prop1 = param1;  this.prop2 = param2;  this.meth1 = function () {  instruction(s);  };  }  var monObjet = new MonObjet(valeurProp1, valeurProp2); |  |  | le mot new permet de créer une instance (cad une copie) de l'objet à partir du constructeur (la fonction qui définit l'objet)  Le nouvel objet aura ses propres caractéristiques puisque les valeurs de ses prop. sont attribuées lors de l'instanciation moment-là |
| instanceof | permet de tester si un objet est une instance (= a été créé à partir) du constructeur | monObjet instanceof MonObjet; |  | => true si vrai  => false si faux |  |
| Object.create( ) | permet de créer un nouvel objet sur le modèle d’un objet déjà existant qui sera alors considéré comme prototype | var objet2 = Object.create(Objet1); |  |  | ici il s'agit d'une copie conforme de l'objet avec ses prop. mais aussi leurs valeurs |
| this. | remplace le nom de l’objet s’il est utilisé dans le corps de l’objet lui-même |  |  | var ObjetA = {  init: function (valeur1, valeur2, valeur3) {  this.prop1 = valeur1;  this.prop2 = valeur2;  this.prop3 = valeur3;  },  methode1 {  },  methode2 {  }  };  var objetB = Object.create(ObjetA);  objetB.init(valeurProp1, valeurProp2, valeurProp3); | prototype ayant une méthode permettant l’initialisation des objets créés à partir de lui grâce notamment à this. |
|  | accéder à la valeur d’une propriété d’un objet | monObjet.propriete2; |  | monObjet["maPropriété"];  var maVariable = "maPropriété";  monObjet[maVariable];  ATTENTION : ne marche pas !  monObjet[unAutreObjet.prop1];  Il faut faire :  var maVariable = unAutreObjet.prop1;  monObjet[maVariable]; | autre syntaxe pour accéder à la valeur d'une propriété d'un objet  qui peut être utile si le nom de la prop. est stocké dans une variable  ATTENTION : cette méthode ne marche qu'avec une chaine directe ! |
|  | modifier la valeur d’une propriété d’un objet | monObjet.propriete2 = nouvelleValeurProp2 |  | monObjet[propriete2] = nouvelleValeurProp2 | idem |
|  | pour créer une nouvelle propriété dans un objet existant en lui attribuant une valeur | monObjet.proprieteN+1 = valeurPropN+1 |  | monObjet[proprieteN+1] = valeurPropN+1 | idem |
| : { } | définition d'une propriété avec toutes ses caractéristiques  (valeurs par défauts : true) | maPropriete: { writable: true, configurable: true, enumerable: true, value: 'maValeur' } |  | writable = modifiable (false => constante)  configurable = accès (false => avec getter et setter)  enumerable = scope (false => accès uniquement dans l'objet.)  equiv "private" en Java, pas accés à la prop par for( in ) | |
| delete | suppression d'une propriété d'un objet | delete monObjet['maPropriete']; |  |  |  |
| function() { } | pour créer une méthode (= fonction dans un objet) dans un objet | var monObjet = {  props: valeursProps,  methode1: function () {  instructions;  },  methode2: function () {  instruction;  }  }; |  |  |  |
|  | appeler la méthode d’un objet | monObjet.methode2(); |  |  |  |
| .prototype | pour créer une méthode au sein du prototype d'un objet | MonObjet.prototype.maMethode = function (param1, param2) {  this.instuction;  instruction(s);  } | *voir* ***Cours-JS-Web2*** *p.6* |  | ATTENTION : cette méthode ne fonctionne qu'appliquée à un constructeur (et non pas à un objet)  Une fois la nouvelle méthode donnée au constructeur, elle sera accessible à tous les objets ayant été construit avec |
| .\_\_proto | accès à l'objet prototype de monObjet | monPrototype = monObjet.\_\_proto; |  |  |  |
| for ( in ) {  } | parcourir les propriétés et méthodes d'un objet une à une | for (var id in monObjet) {  console.log(id);  console.log(monObjet[id];  } | *voir* ***Cours-JS-Web2*** *p.6* |  | Remarque :  le 1er .log affiche les noms des propriétés et méthodes  le 2nd affiche la valeur des propriétés et "fonction" pour les méthodes |
|  |  |  |  |  |  |
| Contexte de méthodes | méthodes permettant de rediriger la référence de this de la méthode qui la précède sur l'objet donné en paramètre |  |  |  |  |
| .apply( ) | prend un tableau en param. facultatif | maMethode.apply(objetRef);  maMethode.apply(objetRef, [valeur1, valeur2]); |  |  |  |
| .call( ) | prend une/des valeurs en param. facultatif | maMethode.call(objetRef);  maMethode.call(objetRef, valeur1, valeur2); |  |  |  |
|  |  |  |  |  |  |
| Map |  |  |  |  |  |
| Map | création d'un objet Map, association de type clé-valeur | var map = new Map();  map.set(cle, valeur);  var maValeur = map.get(cle); |  | *fusion de deux maps :*  var bigMap = new Map([...map1, ...map2]); |  |
| .has( ) | renvoie un booléen indiquant si une valeur associée à cette clé a été trouvée dans l'objet | if(map.has(cle)) {  …  } |  |  |  |
| .size | obtenir la taille de la map | var taille = map.size; |  |  |  |
| .delete( ) | supprimer la paire cle-valeur (retourne true si trouvée et supprimée, false si non trouvée) | map.delete(cle); |  |  |  |
| .clear() | supprime toutes les paires de clé-valeur de l'objet | map.clear(); |  |  |  |
| for( of ) { | boucle for avec une Map | for(var cle of map.keys()) {  console.log(cle);  }  for(var valeur of map.values()) {  console.log(valeur);  } | | for(var [cle, valeur] of map) {  console.log(cle + ' = ' + valeur);  }  map.forEach(function(valeur, cle)); |  |
| Array | création d'une map à partir d'un tableau clé-valeur | var tableauCleValeur = [[clé1, valeur1], [clé2, valeur2]];  var map = new Map(tableauCleValeur); | |  |  |
| créer un tableau clé-valeur à partir d'une map | var tableauCleValeur = Array.from(**map**); | |  |  |
| extraire les clés ou les valeurs sous forme d'un tableau | var tableauCle = Array.from(map**.keys()**);  var tableauValeur = Array.from(map**.values()**); | |  |  |
|  |  |  |  |  |  |
| Temps |  |  |  |  |  |
| new Date(); | création d'une date fixe pouvant servir de référence | maDate = new Date();  maDate.setFullYear(1989);  maDate.setFullYear('1989');  … |  | maDate = new Date(monTimestamp);  maDate = new Date('01/31/1989');  maDate = new Date('01/31/1989 13:58:35');  Date(year, month, day, hours, minutes, seconds, milliseconds); |  |
| .getTime(); | obtenir le timestamp de la date | maDate.getTime(); |  |  | Rq: timestamp = nb de millisecondes écoulées depuis le 1er Jan 1970 |
| .now() | obtenir le timestamp instantané | var now = Date.now(); |  | new Date().getTime(); |
| .getMiliseconds() | obtenir les milisecondes (de 0 à 999) |  |  |  |  |
| .getSeconds() | obtenir les secondes (de 0 à 59) |  |  |  |  |
| .getMinutes() | obtenir les minutes (de 0 à 59) |  |  |  |  |
| .getHours() | obtenir l'heure (de 0 à 23) |  |  |  |  |
| .getDate() | obtenir le jour du mois (de 1 à 31) | maDate.getDate(); |  |  |  |
| .getDay() | obtenir le jour de la semaine (de 0 pour Dimanche à 6 pour Samedi) | maDate.getDay(); |  |  |  |
| .getMonth() | obtenir le mois (de 0 pour Janvier à 11 pour Decembre) | maDate.getMonth(); |  |  |  |
| .getFullYear() | obtenir l'année (au format yyyy) | maDate.getFullYear(); |  |  |  |
| .toLocaleTimeString() | obtenir l'heure au format hh:mm:ss | maDate.toLocaleTimeString(); |  |  |  |
| .toLocaleDateString() | obtenir la date au format dd/mm/yyyy | maDate.toLocaleDateString(); |  |  |  |
|  |  |  |  |  |  |
| Math |  |  |  |  |  |
| % | donne le reste de la division euclidienne | nb1 % nb2 | 10 % 2 => 0  11 % 2 => 1 | if (nb % 2 === 0) { | peut servir à déterminer si un nb est pair ou non. |
|  |  |  |  |  |  |
| Math.min( , , ) | renvoie le minimum d’une liste de nombre | Math.min(nb1, nb2, nb3); |  |  |  |
| Math.max( , , ) | renvoie le maximum d’une liste de nombre | Math.max(nb1, nb2, nb3); |  |  |  |
| Math.random() | renvoie un nombre aléatoire entre 0 et 1 | Math.random() |  |  |  |
| Math.trunc( ) | renvoi la partie entière d'un nombre | Math.trunc(nb); |  |  |  |
| Math.round( ) | renvoi la valeur arrondie d'un nombre | Math.round(nb); |  |  |  |
| Math.PI | valeur de pi |  |  |  |  |
| Math.pow( , ) | calcule la puissance | Math.pow(a, b) | Math.pow(3, 2) => 9  Math.pow(2, 3) => 8 |  |  |
|  |  |  |  |  |  |
|  | Fonctions utilitaires pratiques et réutilisables |  |  | |  |
| roundDecimal( , ) | renvoie un nombre avec N chiffres après la virgule | roundDecimal(monNombre, N); | function roundDecimal(nombre, precision){  var precision = precision || 2;  var tmp = Math.pow(10, precision);  return Math.round( nombre\*tmp )/tmp;  } | |  |
| rand( , ) | retourne un nombre entier aléatoire   * entre min et max inclus si 2 paramètres * entre -min et +min si un seul paramètre | rand(0, 50);  *=> renverra un nb entre 0 et 50*  rand(10, 50);  *=> renverra un nb entre 10 et 50*  rand(-10, 50);  *=> renverra un nb entre -10 et 50*  rand(50);  *=> renverra un nb entre -50 et 50* | /\*\*  \* returns an integer value between min and max  \* or return a value between -min and +min if the second parameter is not specified  \* @param {number} min  \* @param {number} max  \*/  function rand(min, max) {  //if max is not specified then return a value between -min and +min  if (!max) {  max = min;  min = -min;  }  return Math.round(Math.random() \* (max - min) + min);  } | |  |
| moreOrLess( , ) | retourne un nombre entier aléatoire dans un intervalle centrée sur la valeur passée en premier paramètre et +/- le pourcentage indiqué en 2e paramètre | moreOrLess(10, 50);  *=> renverra un nb entre 5 et 15*  moreOrLess(10, 10);  *=> renverra un nb entre 9 et 11*  moreOrLess(50, 50);  *=> renverra un nb entre 25 et 75* | /\*\*  \* returns an integer value between (value - value \* variability) and (value + value \* variability)  \* @param {number} value  \* @param {number between 0 and 100} variability  \*/  function moreOrLess(value, variability) {  return Math.round(rand((100 - variability) \* value, (100 + variability) \* value) / 100)  } | |  |
| randomInTable( ) | retourne un des éléments du tableau au hasard |  | /\*\*  \* returns an element of the array taken randomly  \* @param {array} array  \*/  function randomInTable(array) {  if (array && array instanceof Array && array.length > 0) {  return array[Math.round(Math.random() \* (array.length - 1))];  }  } | |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |